

V2X OBU COMPANION APP

App Requirements & Documentation

**Phase 01 Complete; Phase 02 Active (CAM-Based); Phase 03
Planned (ESP32-C5 & App-Generated CAM)**

Project	MicrOBU
Owner	HAW Hamburg
Platform	Android (Kotlin)
Status	Active

Glossary & Acronyms

All technical terms and acronyms used in this document are defined below for reference.

Term / Acronym	Definition
CAM	Cooperative Awareness Message. A V2X message broadcast autonomously and periodically by all road users (vehicles, cyclists, pedestrians) containing their position, speed, heading, and station type. This is the primary; and currently the only; V2X message type used by the app to detect nearby road users and evaluate safety use cases.
CAMv2	An extended Cooperative Awareness Message format referenced by the CAR 2 CAR Communication Consortium (C2C-CC) Bicycle Safety Use Cases white paper, carrying additional fields relevant to VRU-specific use cases (e.g. finer-grained vehicle class and dimension data) beyond the baseline ETSI EN 302 637-2 CAM. Treated equivalently to CAM for this project's purposes.
DENM	Decentralized Environmental Notification Message. A V2X event message used to warn other road users of hazards; accidents, roadworks, emergency vehicles, weather conditions. Triggered by a specific event and sent to a defined geographic area. Not used by this project ; the app's use cases are built entirely on CAM. Retained here for terminological completeness and because other road users' OBUs may still emit DENM, which the app does not act on.
C2C-CC	CAR 2 CAR Communication Consortium. An industry-driven, non-commercial association founded in 2002 by vehicle manufacturers to standardise cooperative road traffic based on V2V and V2I communication. Source of the White Paper on Bicycle Safety Use Cases (C2CCC_WP_2324, v1.0, 2026-05-07) that underpins Section 1.2 of this document.
BSP	Basic System Profile. The C2C-CC's core interoperability specification (referenced version 1.6.8) defining baseline requirements for V2X stations, including the bicycle-related profile requirements that the C2C-CC bicycle safety use cases build on.
IMA-B / IMA-S	Intersection Movement Assist involving bikes / with Standstill Vehicle. C2C-CC use cases (UC_BIKE_00001-3) in which a bike and car crossing paths at an intersection are mutually alerted via CAM exchange. IMA-B is the project's primary Use Case 1 (Section 1.1).
RTW-B / LTW-B	Right-turn / Left-turn Warning for bike. C2C-CC use cases (UC_BIKE_00004-7) alerting a car driver turning right or left of a bike travelling straight and parallel, via CAM exchange.
SMVA / BCW-B	Slow Moving Vehicle Ahead / Backward Collision Warning for bike. C2C-CC use cases (UC_BIKE_00008-9) alerting a car approaching a slower bike (or peloton) from behind, and alerting the cyclist of a fast-approaching car from behind, via CAM exchange.

BAW	Bike Accident Warning. A C2C-CC use case (UC_BIKE_00010) in which a fallen cyclist's OBU sends a DENM to warn approaching vehicles. Out of scope for this project since it requires DENM generation, which this project does not implement.
MAPEM	MAP Extended Message. A V2X message describing the physical topology of an intersection: lane positions, widths, connections, and permitted manoeuvres. Published by road-side infrastructure (RSUs).
SPATEM	Signal Phase and Timing Extended Message. A V2X message carrying the current and predicted traffic light phases for each signal group at an intersection. Combined with MAPEM data to give in-vehicle signal phase information.
CPM	Collective Perception Message. A V2X message sent by infrastructure or sensor-equipped vehicles to share detected objects (pedestrians, cyclists, vehicles) with other road users who may not have line of sight.
SREM	Signal Request Message. A V2X message sent by a vehicle (typically public transport or emergency services) to an intersection to request signal priority or preemption.
SSEM	Signal Status Message. A V2X message sent by an intersection in response to an SREM, indicating whether a signal priority request has been granted, rejected, or is being processed.
IVIM	In-Vehicle Information Message. A V2X message used to convey road operator information; speed limits, road signs, advisories; in a machine-readable format.
V2X	Vehicle to Everything. The umbrella term for wireless communication between a road user and any other entity: other vehicles (V2V), infrastructure (V2I), pedestrians (V2P), and networks (V2N). In Europe, implemented via the ITS-G5 standard at 5.9 GHz.
ITS-G5	Intelligent Transport Systems at 5.9 GHz. The European standard (based on IEEE 802.11p / DSRC) for short-range, low-latency V2X communication. Used by the microOBU for direct peer-to-peer messaging without a cellular network.
C-ITS	Cooperative Intelligent Transport Systems. Systems in which vehicles and infrastructure share real-time information to improve safety, traffic flow, and efficiency. The broader ecosystem in which V2X operates.
DSRC	Dedicated Short Range Communication. The wireless technology (IEEE 802.11p) underlying ITS-G5. Provides low-latency direct communication between vehicles and infrastructure within approximately 300–1000 m range.
OBU	On-Board Unit. The physical V2X hardware device. Phase 02 uses the consider it CiT One OBU, a compact autonomous V2X unit. Phase 03 introduces the ESP32-C5 as a second, selectable OBU option acting as a dumb ITS-G5 transceiver (Section 13); the user selects which OBU is connected in Settings (Section 13.3).

ESP32-C5	A Wi-Fi 6 / Bluetooth 5 / 802.15.4 System-on-Chip from Espressif, being evaluated in Phase 03 as a low-cost, custom "dumb" OBU option: it transmits and receives V2X frames on the phone's instruction rather than running its own autonomous CAM generation logic (Section 13.1). Its ITS-G5 (5.9 GHz) RF/PHY compliance is still to be validated, not assumed.
RSU	Road Side Unit. A fixed V2X infrastructure unit installed at the roadside (typically at intersections) that communicates with passing vehicles and VRUs. RSUs publish MAPEM, SPATEM, and CPM messages.
VRU	Vulnerable Road User. Any road user not protected by a vehicle shell: pedestrians, cyclists, e-bike riders, motorcyclists. VRU protection is the primary application scenario for the microOBU.
HMI	Human-Machine Interface. Any interface through which a user interacts with or receives information from a system. The Android companion app serves as the HMI for the microOBU, displaying V2X alerts, connection status, and sensor data.
GNSS	Global Navigation Satellite System. The collective term for satellite-based positioning systems (GPS, GLONASS, Galileo, BeiDou). Used by both the microOBU hardware and the smartphone to determine position, speed, and heading.
IMU	Inertial Measurement Unit. A sensor combining accelerometers and gyroscopes to measure acceleration and angular velocity. The smartphone's IMU provides motion data for dead reckoning, orientation detection, and OBU antenna coordination.
MQTT	Message Queuing Telemetry Transport. A lightweight publish-subscribe messaging protocol. All communication between the microOBU and the Android app passes through an MQTT broker (Mosquitto 2.0.11) running on the OBU, accessible on port 1883.
BLE	Bluetooth Low Energy. A power-efficient variant of Bluetooth optimised for short bursts of data. The planned Phase 03 production transport between the microOBU and the Android phone, with a target connection setup time of ≤ 2 s.
MVVM	Model-View-ViewModel. The Android architecture pattern used throughout the app. The Model (Repository and data sources) is decoupled from the View (Jetpack Compose UI) by a ViewModel that holds and exposes UI state via StateFlow and SharedFlow.
PKI	Public Key Infrastructure. A system of cryptographic certificates and keys used to authenticate and sign communications. The microOBU uses signed V2X messages; the app is responsible for storing and using OBU public keys to verify message authenticity.
HAW	Hamburg University of Applied Sciences (Hochschule für Angewandte Wissenschaften Hamburg). Project owner and research partner, responsible for the microOBU project coordination and application development.

0. Project Context & Main Goal

The MicroBU project aims to develop a compact, lightweight on-board unit for vulnerable road users (VRUs); such as cyclists, e-bike riders, and pedestrians; enabling them to participate actively in the V2X (Vehicle-to-Everything) network and thereby improve their safety in traffic.

0.1 Technological Innovation Context (from Project Overview Document CiT/HAW)

The microBU project represents a significant level of innovation across four domains. The Android companion app is a direct expression of the Software and Integration pillars, and its requirements must be understood in that context.

Domain	Key Innovations & App Implications
A) Architecture	Novel resource-sharing architecture between microBU and the smartphone (host system). The app must implement cybersecurity measures including verification of signed OBU communications using public key cryptography.
B) Hardware	Compact OBU achieved through novel antenna integration and shared carrier hardware. Low weight via minimised battery. Scalable, modular hardware architecture. The app must gracefully support different OBU hardware configurations and firmware versions.
C) Software	Intelligent, adaptive distribution of processing tasks between microBU and the smartphone. Dynamic adjustment of OBU antenna characteristics and communication frequency based on traffic conditions, movement, and orientation; coordinated via the app. Application for integrating the carrier system (smartphone) with the microBU.
D) Integration & Application	Seamless connection to the microBU via standardised existing interfaces (Bluetooth, USB). Application scenarios target micromobility users (e-bike riders, cyclists) and pedestrians.

0.2 Bicycle Safety Use Case Foundation (CAR 2 CAR Communication Consortium White Paper)

The project's bike-car safety use cases are grounded in the CAR 2 CAR Communication Consortium's White Paper on Bicycle Safety Use Cases (C2CCC_WP_2324, v1.0, 2026-05-07) [14]. The white paper analyses bicycle-to-car accident statistics (GIDAS, CATS) and defines six standardised V2V/V2I use case families addressing the most common and most severe crash scenarios; intersection crossing, turning conflicts, and same-direction closing speed. Full details and the mapping to this project's scope are given in Section 1.2.

Critically for this project's technical direction: five of the six C2C-CC use cases (IMA-B, IMA-S, RTW-B, LTW-B, SMVA/BCW-B) are achievable using **CAM / CAMv2 exchange alone**, requiring no event-triggered messaging. Only the sixth use case, Bike Accident Warning (BAW), depends on DENM. Since this project uses **CAM as its primary; and only; V2X**

message type, BAW is explicitly out of scope, while the other five directly inform the app's detection and alerting requirements.

Main Goal of the Android App

Core Purpose

- Serve as the primary user interface and intelligent processing hub of the V2X OBU
- Bridge communication between the V2X hardware module and the user
- Minimize power consumption while maximizing situational awareness
- Ensure seamless integration into the user's existing mobile ecosystem

Four Core Pillars

Pillar	Description
Safety & Real-Time Alerting	Receive, process, and present CAM messages from the OBU hardware (own and remote road users) and translate them into timely warnings; audio, haptic, or visual; using the three-tier C2C-CC alert-level model (Info / Awareness / Warning, Section 5.5).
Offloading OBU Complexity	Leverage the smartphone's processing power, GPS, IMU, and connectivity to reduce the hardware burden on the physical OBU, keeping it compact and lightweight.
Power-Aware Operation	Operate efficiently in the background with minimal battery drain on both phone and OBU, supporting extended sessions during cycling or riding.
Seamless User Experience	Require minimal interaction during use; intelligent background operation with simple setup, so the user can focus entirely on traffic.

1. Phase 01 Overview; Data Collection

Phase 01 is about building the data foundation of the app: establishing reliable connections to the OBU hardware, harvesting all relevant sensor streams, and providing tools to capture and inspect real-world data for later analysis and algorithm development.

1.1 Test Intersections & Use Cases (Testkreuzung)

Based on exchanges with LSBG and HHVA, as well as input from local cyclists, the following intersections have been identified as high-risk traffic accidents and will serve as primary test locations:

Intersection	Status
Kreuzung 1	To be defined; high accident risk confirmed by stakeholders
Kreuzung 2	To be defined; high accident risk confirmed by stakeholders

To deliver real-world value, use cases will be tested either directly at these intersections or in a simulated equivalent of the intersection scenario.

Use Case 1; Mikromobilität (Micro-Mobility); C2C-CC Intersection Movement Assist for bikes (IMA-B)

Use Case Details

- Owner: HAW Hamburg
- C2C-CC reference: Intersection Movement Assist involving bikes (IMA-B), UC_BIKE_00001 / UC_BIKE_00002 [14]
- Primary Actor: E-bike rider
- Secondary Actor: Car driver
- Relevant V2X Message: **CAM only** (Cooperative Awareness Message); no DENM is used or required
- Goal: The car driver detects the e-bike rider via an HMI. The e-bike rider can optionally also detect the car driver via an HMI.

Use Case 1; Flow

- Car driver attempts to turn right at the intersection.
- E-bike rider travels along a real or simulated road and crosses the street into which the driver is turning.
- The HMI of both the car driver and the e-bike rider detects the other road user via CAM messages exchanged directly between their OBUs (no RSU required for the V2V variant).

Phase 01 data collection at or near these intersections will directly feed into the CAM-based detection and alerting logic required for this use case.

1.2 C2C-CC Bicycle Safety Use Case Mapping & Roadmap

The table below maps all six use cases defined in the C2C-CC White Paper on Bicycle Safety Use Cases [14] to this project's scope, message dependency, and C2C-CC alert level (see Section 5.5 for the alert-level model itself).

Use Case	C2C-CC ID	C-ITS Message	Project Scope
Intersection Movement Assist for bikes (IMA-B)	UC_BIKE_00001/2	CAM, CAMv2	In scope; Use Case 1, primary project use case (Section 1.1)
Intersection Movement Assist with Standstill Vehicle (IMA-S)	UC_BIKE_00003	CAM, CAMv2	In scope for future phase; alerts a stopped car driver of an approaching bike as the car starts moving or turning
Right-turn Warning for bike (RTW-B)	UC_BIKE_00004/5	CAM, CAMv2	In scope for future phase; highest accident relevance per GIDAS analysis (UTYP #342)
Left-turn Warning for bike (LTW-B)	UC_BIKE_00006/7	CAM, CAMv2	In scope for future phase; same mechanics as RTW-B, mirrored geometry
Slow Moving Vehicle Ahead (SMVA) / Backward Collision Warning for bike (BCW-B)	UC_BIKE_00008/9	CAM, CAMv2	In scope for future phase; relevant on rural/same-direction roads, dominant in lateral-accident statistics (CATS)
Bike Accident Warning (BAW)	UC_BIKE_00010	CAM (own) + DENM (alert)	Out of scope; depends on DENM generation, which this project does not implement

Accident-statistics rationale (from the white paper's Appendix A, based on GIDAS and CATS data): crossing scenarios (UTYP #341/#342, addressed by IMA-B and RTW-B) are the most probable bike-car collision types, especially where visual obstruction is present; lateral same-direction accidents (addressed by SMVA/BCW-B) are dominant in several EU regions and in the US and carry a high share of severe/fatal injuries. This prioritisation supports keeping IMA-B as Use Case 1 for Phase 01/02 and treating RTW-B, LTW-B, and SMVA/BCW-B as the next logical extensions, all achievable with the CAM-only architecture already being built.

2. Functional Requirements

2.1 Screens & Features

Screen	Description
Dashboard / Home	Live overview of all active data streams; connection status, GNSS fix quality, IMU activity, MQTT broker status, and active topics at a glance. Animated MQTT connection-state indicator throughout the top bar and dashboard reflecting real broker state: Disconnected / Connecting / Connected / Error. All state indicators update in real time from the live MQTT connection. Phase 03 addition: redesigned into sensor overview cards (GNSS, IMU, OBU connection status) plus a prominent "Start Driving Session" shortcut that navigates directly to the Recording Session screen (Section 13.7).
Connection Setup	Connect to OBU via USB-C (primary for Phase 02) or Bluetooth (Phase 03 production target). Wi-Fi toggle visible only in developer builds; hidden in production releases. USB-C connection is initiated via Android USB tethering; once plugged in, the app detects the USB network interface and connects to the MQTT broker at the OBU's tethering IP automatically. Phase 03 addition: USB-C is also the transport when the connected OBU is the ESP32-C5, but over a UART serial link rather than USB Ethernet tethering (Section 13.2); which OBU is connected is chosen in Settings (Section 13.3).
Sensor Monitor	Real-time readout of all active phone sensors (GNSS, IMU, barometer, etc.) and OBU GNSS side by side for cross-reference.
V2X Monitor	Live stream of all V2X messages received from the OBU, with primary focus on CAM (remote road users) and the OBU's own state. Messages are grouped by MQTT topic with pretty-printed JSON payload, arrival timestamp, and message counter per topic. An auto-scroll toggle keeps the latest message in view or freezes the list for inspection. Subscribes to OBU topics on connection: sys/state/heartbeat, sys/state/cellular, v2x/rx/obu_gnss (ego bike's own position/speed/heading/stationType), v2x-uca/output/json/cam (CAMs received from nearby remote vehicles/bikes), v2x-uca/output/json/spat, v2x-uca/output/json/map, v2x-uca/output/json/cpm. In Phase 02 the broker is reached over USB-C tethering (primary) or Wi-Fi (dev mode). Phase 02 adds a CAM-based Use Case Alert panel: the app continuously evaluates incoming remote CAMs against the ego bike's own state (from obu_gnss) to detect C2C-CC bike safety use case conditions (IMA-B, and in future phases RTW-B, LTW-B, IMA-S, SMVA/BCW-B) and surfaces the resulting alert level (Info / Awareness / Warning, Section 5.5). Phase 03 addition: a map view (Section 13.7) plots the ego bike's own GNSS position and detected remote vehicles' positions from CAM data, as an alternative to reading raw per-topic JSON.
Recording Session	Start/stop data capture across all streams simultaneously. Every sensor reading (accelerometer, gyroscope, magnetometer, barometer, GNSS) is written as a timestamped CSV row in real time to internal storage during the session. Persistent, always-visible recording state indicator and live session duration counter throughout the app. Phase 03 addition: starting a recording session is also what enables phone-generated CAM transmission via the

	ESP32-C5 (Section 13.5, Section 13.6).
Session Log / Export	Browse past recording sessions, preview data, export as timestamped CSV. Share via the standard Android share sheet.
Settings	Navigable settings menu: tapping a category opens a dedicated sub-screen rather than scrolling a single long page. Categories: Language, Appearance, Sensors, Units, MQTT Broker (IP, port, username, password; persisted via DataStore), Use Case Alerts, Developer, and About. MQTT broker settings take effect on the next connection attempt without restarting the app. Phase 03 addition: an OBU Hardware category to select CiT One or ESP32-C5 (Section 13.3).

2.2 User Interactions

- One-tap start/stop recording with a persistent recording status indicator
- Manual MQTT topic subscription management (add/remove/filter)
- Bluetooth device pairing and connection management
- Per-sensor enable/disable toggles
- Per-use-case alert enable/disable toggles (IMA-B now; RTW-B, LTW-B, IMA-S, SMVA/BCW-B as they come into scope)
- CSV export and sharing from session history

2.3 Backend, Database & Login

Concern	Decision
Remote backend	None required for Phase 01; fully local and self-contained
User login / authentication	Not required for Phase 01
Local database	Room; for session metadata and file references
File storage	Local app-specific storage (scoped storage / MediaStore) for CSV exports
MQTT broker	Runs on the OBU hardware (Mosquitto 2.0.11, port 1883); the app is purely an MQTT client. Phase 01 (complete): broker reachable at 192.168.3.202 over Wi-Fi. Phase 02 (active): USB-C tethering; OBU accessible at Android tethering subnet IP (typically 192.168.42.x). Phase 03 (production): Bluetooth transport.

3. Non-Functional Requirements

3.1 Android Version & Target Devices

Parameter	Value
Min SDK	29 (Android 10); ensures modern Bluetooth APIs, scoped storage, and foreground service behaviour
Target SDK	35 (Android 15); latest stable target, Play Store compliant
Primary devices	Smartphones; portrait and landscape
Secondary devices	Tablets; adaptive layout desirable but not a Phase 01 priority
Foldables	Out of scope for Phase 01

3.2 Performance & Offline

- Sensor polling and MQTT message handling must run in a background foreground service; minimal data loss on screen lock or app backgrounding
- GNSS + IMU + MQTT simultaneous streaming with minimal perceptible UI lag
- CSV export of a full session must be completed within acceptable time for typical session lengths
- Graceful degradation if a sensor or connection is unavailable; app continues with remaining active streams
- Fully offline by design; no internet connection required or used
- All data always remains on device

4. Technical Requirements

4.1 Development Environment (from Google Dev Docs)

Tool	Detail
IDE	Android Studio Panda 4; current stable release
Language	Kotlin; primary language throughout
Build system	Gradle with Version Catalogs (libs.versions.toml)
Version control	Git; feature/* / develop / main branching strategy

4.2 Architecture

- MVVM + Repository pattern with ViewModel, StateFlow / SharedFlow
- Clean separation between data sources and UI
- Jetpack Compose for all UI; no XML layouts
- Intelligent, adaptive task distribution: computationally intensive processing (position fusion, CAM-based use case detection) is offloaded from the microOBU to the smartphone, keeping the hardware compact and power-efficient
- **CAM-based Use Case Detection Engine:** maintains a rolling per-station history of recent remote CAMs (keyed by stationId, from v2x-uca/output/json/cam) and the ego bike's own recent state (from v2x/rx/obu_gnss), and correlates the two to evaluate the geometric and kinematic conditions for each C2C-CC bike safety use case in scope (closing paths, relative heading trend, distance to conflict point, closing speed derived from successive samples) and raises alerts per the three-tier alert model (Section 5.5). No DENM generation or consumption is part of this engine.
- OBU antenna coordination: the app relays configuration commands to the microOBU to dynamically adjust antenna directionality and V2X communication frequency based on traffic conditions, device speed, and orientation (derived from IMU and GNSS)
- MQTT reconnection: automatic exponential-backoff reconnection to the broker on connection loss. The client retries with increasing delays (1 s, 2 s, 4 s ... up to a configurable ceiling) until the broker is reachable again, without user intervention

4.3 Key Libraries (from Google Developer Docs)

Category	Library	Purpose
UI	Jetpack Compose (BOM) + Material3	All UI components and theming
Navigation	Compose Navigation	Screen routing and back-stack
Async / Concurrency	Kotlin Coroutines + Flow	Reactive data streams and async ops
Bluetooth	Android Bluetooth APIs	Classic and BLE connection to OBU

MQTT	Eclipse Paho Android Client (org.eclipse.paho:org.eclipse.paho.android.service)	V2X topic subscription and messaging
JSON / CAM parsing	kotlinx.serialization	Parsing processed CAM JSON from v2x- uca/output/json/cam into typed models for the Use Case Detection Engine
Local Database	Room	Session metadata and file references
Sensor Access	Android SensorManager	IMU, barometer, via repository layer
GNSS	FusedLocationProviderClient	Phone GNSS position and speed
CSV Export	OpenCSV / Custom Kotlin writer	Timestamped session data export
Dependency Injection	Hilt	DI framework throughout the app
Permissions	Compose-native permission handling	Runtime permission management
Logging (dev)	Timber	Debug and development logging

4.4 Security Requirements (future)

The microOBU project mandates cybersecurity as a core system property. Communications between the OBU hardware and the companion app must be signed and verifiable. The app is responsible for holding and managing the public keys used to authenticate OBU-originated messages, ensuring that safety-critical V2X data (CAM) cannot be spoofed or tampered with in transit.

Requirement	Detail & Implementation Note
Signed OBU message verification	All CAM messages received from the OBU must carry a digital signature. The app verifies signatures using the OBU's public key before processing safety-critical data. Use Android Keystore API for secure key storage; BouncyCastle (already bundled via MQTT client) for ECDSA/RSA verification.
Public key provisioning	The Settings screen must expose a key management flow: import OBU public key (QR code or file), inspect current trusted keys, and revoke/replace keys. Keys stored in Android Keystore or encrypted SharedPreferences. Exact protocol provisioning to be defined in collaboration with HAW Hamburg.
Failure handling for invalid	CAM messages that fail signature verification must be silently dropped and logged (never surfaced as safety alerts). A persistent warning

signatures	indicator must be shown in the Dashboard if verification failures exceed a configurable threshold, signalling a potential security event.
No internet security dependency	The fully offline design (Section 3.2) means all cryptographic operations run on-device with no reliance on online certificate revocation or cloud-based trust services. Public key trust anchors must be manually provisioned and managed within the app.

5. Design Requirements (to-do)

5.1 Visual Language

- Material Design 3 throughout; dynamic color, updated typography scale, M3 components
- Dark mode first; the app is frequently used outdoors and glanced at while in motion; dark UI reduces glare and improves readability
- Light mode supported as a secondary, switchable theme

5.2 System Bar & Edge-to-Edge

- Full edge-to-edge display enforced (`WindowCompat.setDecorFitsSystemWindows = false`)
- Proper inset handling via `windowInsetsPadding` in Compose for all screens
- Status bar and navigation bar styled to be transparent or appropriately coloured per screen

5.3 UX Principles

Principle	Description
Glanceability	Critical status info (connection state, recording status, GNSS fix quality) must be readable at a glance, even while in motion.
Minimal interaction during use	The app should largely run itself once a session is started; the user should not need to interact with it while cycling or riding.
Clear feedback	Every connection state, error, or data gap must be visually communicated; never silent. Use colour, iconography, and short text labels.
Low friction setup	Onboarding and initial connection should be (ideally) completable in under one minute.

5.4 Wireframes & Mockups

Wireframes and high-fidelity mockups are to be produced as a next step. Recommended tool: Figma with the Material Design 3 Figma kit for fidelity and handoff consistency.

5.5 Alert Level Model (C2C-CC Three-Tier Model)

All CAM-based use case alerts (Section 1.2, Section 10.2) are surfaced using the three-tier alert level model defined in the C2C-CC White Paper on Bicycle Safety Use Cases [14]. This model governs both the visual/audio design of the HMI and the timing thresholds the detection engine targets.

Alert Level	Definition	Example Time-to-Conflict Threshold
Info	Lowest severity. Continuous, non-aggressive visualisation; an optional temporal audio cue. Danger is present but not imminent.	> 10–30 s

Awareness	Higher severity; does not require lane-level positioning accuracy to trigger. Continuous visualisation supported by audio signals, timed to give the rider/driver sufficient time to assess and react. This is the level achievable with current BSP1-compliant CAM accuracy.	> 3–15 s
Warning	Highest severity. Continuous, aggressive audio-visual signal. Indicates imminent danger. Full lane-level position accuracy raises use cases from Awareness to Warning where achievable.	≤ 5 s

Per the white paper's use-case-to-alert-level mapping, all in-scope use cases (IMA-B, IMA-S, RTW-B, LTW-B, SMVA/BCW-B) are expected to reach **Awareness** level under current BSP1-compliant CAM accuracy, with **Warning** level achievable once at least lane-level positioning accuracy is available. The app's HMI and alert engine should be designed to support both levels from the start, upgrading automatically as positioning accuracy improves.

6. Sensor Data Streams; Phase 01

Source	Data / Parameters
Phone GNSS	Position (lat/lon), speed, heading, accuracy, fix quality
OBU GNSS	Position (lat/lon), speed, heading; for cross-reference and redundancy
Phone IMU; Accelerometer	3-axis acceleration (m/s ²)
Phone IMU; Gyroscope	3-axis angular velocity (rad/s)
Phone IMU; Magnetometer	3-axis magnetic field (heading reference)
Phone Barometer	Atmospheric pressure, derived altitude
MQTT Topics (OBU)	CAM (primary; remote road users via v2x-uca/output/json/cam, ego bike's own state via v2x/rx/obu_gnss), CPM, SPAT, MAP; all fully configurable. DENM is not consumed or acted upon in the current project scope.
Additional phone sensors	Any further sensors identified as relevant during testing (e.g. ambient light, proximity)

7. Recording Session & Data Export

7.1 Recording Mode

- Dedicated Recording Session mode; user manually starts and stops capture
- All active sensor streams and MQTT messages captured simultaneously with synchronised timestamps
- Persistent foreground service ensures minimal data loss during screen lock or backgrounding
- Clearly visible recording state indicator and live session duration counter throughout the app

7.2 CSV Export Format

- One structured CSV file per session written in real time; every sensor sample is appended as it arrives, not buffered post-session. This ensures data is preserved even if the session ends abnormally.
- Unified 13-column schema for all sensor types (type, timestamp_ms, timestamp_iso, lat, lon, alt_m, speed_ms, bearing_deg, accuracy_m, x, y, z, pressure_hpa). Columns not applicable to a given sensor type are left empty. Rows include both Unix epoch (ms) and ISO 8601 timestamps.
- Session metadata header: date, duration, device model, app version, OBU firmware version (if available)
- Sessions stored locally in app-scoped storage
- Shareable via standard Android share sheet (e.g. to Files, Drive, email)

8. Connectivity

The app uses a phased connectivity strategy. Phase 01 (complete) used Wi-Fi to reach the OBU's MQTT broker in a lab environment. Phase 02 (active) uses USB-C as the primary transport; the physical cable connecting phone to OBU. Phase 03 (production) will use Bluetooth BLE. The architecture is transport-agnostic: the MQTT client, topic subscriptions, and all UI layers remain identical across all phases; only the underlying network path changes.

8.1 Connection Phase Roadmap

Phase	Transport	Status	Notes
Phase 01	Wi-Fi / Ethernet	Active Now	Dev lab setup. MQTT broker at 192.168.3.202 (configurable). App connects directly over the local network; no physical OBU hardware required during this phase. Credentials stored in Settings. Goal: get MQTT topic display working end-to-end on the app. Wi-Fi visible in all builds; will be hidden in production release (see Section 8.2).
Phase 02	USB-C (Wired)	Active Now	USB-C is the primary physical transport. Android USB tethering mode exposes the OBU as a USB Ethernet network interface; the app resolves the OBU's IP on the tethering subnet and connects to the MQTT broker (port 1883). Phase 02 goal: receive and correctly parse CAM messages from both the OBU's own generated CAM and any nearby remote vehicles' CAMs on v2x-ucca/output/json/cam, and verify the CAM-based Use Case Detection Engine (Section 10.2) correctly raises an IMA-B alert. Wi-Fi remains available in developer builds only.
Phase 03	USB-C (UART, ESP32-C5) + Bluetooth (open discussion)	Planned	Introduces the ESP32-C5 as a second, selectable OBU option (Section 13) alongside the CiT One: a "dumb" transceiver that transmits phone-generated CAM and receives V2X frames on the phone's behalf, connected via USB-C to the ESP32's UART port. Bluetooth as a production transport (BLE target connection setup time ≤ 2 s, per project spec) is back under discussion now that ESP32-C5 is in play, but not yet decided (Section 13.9). See Section 13 for full detail.

8.2 Transport Methods Detail (according to the Project Overview Document)

Method	Role & Notes
Bluetooth (Classic / BLE)	Primary connection to the OBU. Low power, wireless, and practical for a

	mounted or wearable unit. Default in all builds.
USB-C	Primary transport for Phase 02. The phone connects to the OBU via USB-C cable; the phone's Android USB Tethering mode creates a virtual Ethernet interface through which the OBU's MQTT broker (port 1883) is reachable. The app detects the active USB network interface and auto-populates the broker IP. Provides a stable, low-latency, wired link for CAM reception testing and bench development. Hidden in production builds.
Wi-Fi	Developer / legacy transport only. Phase 01 used Wi-Fi to reach the dev broker at 192.168.3.202. From Phase 02 onward, Wi-Fi is retained in developer builds as a fallback and for regression testing, but is not used in normal operation. Hidden in all production/release builds to avoid interfering with the phone's normal network functionality. IP, port, and credentials remain configurable in Settings; Dev section.
MQTT (over BT/USB/Wi-Fi)	App connects to the MQTT broker hosted on the OBU hardware. Configurable broker IP, port, and credentials in Settings.
OBU Antenna Command Channel	Beyond passive MQTT subscription, the app must support a bidirectional command channel to the OBU for relaying antenna configuration updates: dynamically adjust antenna directionality and V2X communication frequency in response to movement speed, heading, and local traffic density derived from received CAM messages. Protocol and MQTT topic schema to be defined with HAW Hamburg.

9. Phase 01 Key Design Principles

Principle	Detail
Modularity	Each data source is an independent, toggleable stream. Individual sensors can be disabled or debugged without affecting others.
Transparency	A live dashboard shows the status and current value of every active data source at a glance.
Logging robustness	Minimal data loss on app backgrounding, screen lock, or brief disconnections. Foreground service handles all sensor collection.
Low friction	One-tap start/stop recording. Clearly visible recordings always. Minimal configuration needed to begin a session.

10. Phase 02 Overview; OBU Communication (CAM-Based Use Case Detection)

Phase 02 marks the first time the Android app communicates directly with the physical microOBU hardware. The primary objective is to establish a reliable wired connection via USB-C and demonstrate end-to-end CAM message flow: the app receives the OBU's own autonomously-generated CAM plus any remote CAMs from nearby vehicles or bikes, and evaluates the C2C-CC bike safety use case conditions (starting with IMA-B, Section 1.2) purely from CAM content. No DENM triggering, transmission, or reception handling is part of this phase or of the project's current scope. Wi-Fi remains available in developer builds only as a regression testing fallback.

10.1 USB-C Wired Transport

Android USB Tethering creates a virtual Ethernet interface over the USB-C cable. When the phone enables USB tethering toward the OBU, the OBU appears on the phone as a network device at a predictable subnet (typically 192.168.42.0/24 on Android). The MQTT broker running on the OBU (Mosquitto 2.0.11, TCP port 1883) is then reachable via this interface. No router or access point is required; the link is point-to-point and fully offline.

Implementation requirements:

- The app detects the active USB Ethernet interface at startup and on USB hot-plug events using ConnectivityManager and NetworkCallback APIs.
- The OBU's IP on the tethering subnet is auto-detected (ARP scan or OBU-specific known offset) and stored. A manual IP override remains available in Settings for edge cases.
- The Connection Setup screen displays a USB-C connection state indicator (Disconnected / Detected / MQTT Connected) in addition to the existing Wi-Fi indicator. USB-C state is derived from the live MQTT connection health, not just cable presence.
- The existing MQTT exponential-backoff reconnection logic (Section 4.2) applies to the USB-C transport without modification. If the cable is removed and re-inserted, the MQTT client reconnects automatically once the USB network interface is restored.
- Wi-Fi transport remains visible and functional in developer builds. In the Settings; Developer section, the user can toggle between USB-C and Wi-Fi transports. Wi-Fi is not shown and not usable in production/release builds.

10.2 CAM Reception & Use Case Detection

A CAM (Cooperative Awareness Message) is defined by ETSI EN 302 637-2 and is broadcast autonomously and periodically by every V2X station, including the microOBU, containing position, speed, heading, station type, and vehicle dimensions. Unlike DENM, no application trigger is required to generate a CAM: the OBU's own V2X stack handles CAM generation and broadcast autonomously, using the OBU's own onboard GNSS-derived kinematics (speed, heading, acceleration, yaw rate, curvature). Per the consider it MQTT API [1], there is no MQTT topic to send, request, or influence CAM content; CAM has no trigger interface at all (unlike DENM, SREM, SSEM, and SPAT, which do). The app's role in Phase 02 is purely to **consume** two separate topics and run the detection logic for the C2C-CC bike safety use cases on top of them. This autonomous-generation behaviour is specific

to the current Phase 02 hardware (the consider it CiT One OBU); Section 13 describes how this inverts in Phase 03, where CAM content generation moves to the phone.

Two distinct data sources feed the detection engine; they are not the same topic:

- `v2x/rx/obu_gnss`: the ego bike's **own** state, published by the OBU itself at 4 Hz. Contains gnss (latitude, longitude, altitude, speed_mps, heading_deg, acceleration_mpss, curvature) and own_info (stationID, stationType, vehicleRole, vehicleWidth_m, vehicleLength_m, yawRate_degps). This is how the app verifies the OBU is correctly configured as stationType = cyclist (2); there is no MQTT way to set it, only to read it back.
- `v2x-uca/output/json/cam`: CAMs **received from other nearby stations** only (cars, other bikes); per the consider it MQTT API documentation, this topic "contains vehicle information received from CAM data." It never contains the ego OBU's own CAM. Each message carries stationId, stationType, position, heading_deg, speed_mps, yawRate_dps, vehicleWidth_m/vehicleLength_m.

CAM-based Use Case Detection data flow:

- The app subscribes to both `v2x/rx/obu_gnss` and `v2x-uca/output/json/cam` on connection.
- The Use Case Detection Engine maintains a short rolling history (not just the single latest message) per remote stationId, and a rolling history of the ego bike's own recent obu_gnss samples. Turning, braking, and closing/receding trends are never flagged directly in a message; they must be derived by comparing successive samples over time, the same way the OBU's own CAM content is itself derived from successive GNSS fixes.
- For each incoming remote CAM, the engine updates that station's history and computes: relative position and distance to the ego bike (from the latest obu_gnss sample), closing speed (from the trend in distance over the last few samples), heading delta between the remote station's heading and the ego bike's heading, and estimated time-to-conflict-point.
- The engine evaluates the geometric and kinematic conditions for each enabled use case (starting with IMA-B; RTW-B, LTW-B, IMA-S, and SMVA/BCW-B in later phases) as defined in Section 1.2.
- When a use case condition is met, the corresponding alert level (Info / Awareness / Warning, Section 5.5) is raised and surfaced via the HMI (audio/haptic/visual, per Section 2 and Pillar 1).
- No outbound V2X message is generated by the app. The OBU's own autonomous CAM broadcast (received by the car or other bike) already carries the ego vehicle's presence; this standard bidirectional CAM exchange is exactly what the C2C-CC use cases rely on.

Required JSON fields consumed by the app:

- From `v2x/rx/obu_gnss` (own state): gnss.latitude/longitude, gnss.speed_mps, gnss.heading_deg, gnss.acceleration_mpss, gnss.curvature, own_info.stationID, own_info.stationType, own_info.yawRate_degps
- From `v2x-uca/output/json/cam` (remote CAMs): stationId, stationType (cyclist = 2, passengerCar = 5, etc., per ETSI EN 302 637-2 Table 1), position, heading_deg, speed_mps, yawRate_dps, vehicleWidth_m / vehicleLength_m (used to refine time-to-conflict estimates where available)

stationType for VRU: the micrOBU's own CAMs must report stationType = cyclist (2) per ETSI EN 302 637-2 Table 1. This is a device-level configuration of the OBU itself, not something set over MQTT by the app; the app can only read it back, via own_info.stationType on `v2x/rx/obu_gnss`. The app should verify this value on connection and alert the user if it is not 2, since this affects VRU-specific handling (e.g. the IMA-B use case) at equipped intersections and vehicles.

10.2.1 Detection Algorithm Detail

This subsection records the concrete algorithm parameters used by the implemented Use Case Detection Engine, as an addendum to the conceptual data flow above. These are

implementation decisions rather than fixed requirements, and may be re-tuned from real ride data; they are captured here for traceability between this document and the app codebase.

- **Own-state source and fallback:** `v2x/rx/obu_gnss` (~4 Hz) is the primary source for the bike's own position, speed, heading, and yaw rate. If `obu_gnss` goes stale for more than 2.5 s, the engine falls back to the phone's GPS for the bike's own position/speed/heading only. Yaw rate is unavailable in this fallback mode, since phone GPS alone does not provide it; use cases that depend on the bike's own turning behaviour degrade gracefully rather than falsely reporting zero rotation. This OBU-primary / phone-fallback pattern is specific to the current CiT One hardware phase; Section 13 describes how Phase 03 (ESP32-C5) inverts this, making the phone's own fused state the primary, always-on source.
- **Per-vehicle rolling history:** each remote station's CAMs are appended to a rolling history of approximately 8 samples / 5 s, keyed by `stationId`. Trends (is heading actually rotating, is distance actually shrinking) are computed from this window rather than from a single incoming sample, to reject single-message noise.
- **Geometry computation:** for each remote vehicle, the engine computes distance and relative bearing to the bike's latest own-state fix, closing speed (from the distance trend across the history window), and a closest-point-of-approach (CPA) projection; the time and distance at closest approach assuming both parties hold their current trajectory.
- **Turning-toward-bike test:** yaw rate (or, if the field is missing from a given CAM, a heading-rate trend derived from the station's history) is used to determine not just that a vehicle is turning, but that the rotation direction is swinging its heading toward the bike's bearing specifically. This is combined with the distance/heading/speed gates to disambiguate the five in-scope use cases: IMA-B (crossing paths), IMA-S (a stopped vehicle about to pull out), RTW-B/LTW-B (a vehicle turning across the bike's path), and SMVA/BCW-B (fast closing speed, same direction).
- **Alert bucketing:** the resulting time-to-conflict estimate is bucketed into Info (≤ 30 s), Awareness (≤ 15 s), or Warning (≤ 5 s), applying the thresholds from Section 5.5 as discrete cutoffs.
- **Clearing conditions:** an active alert clears automatically once the geometry resolves (paths diverge / CPA no longer applies) or once a vehicle goes stale; no CAM received from that `stationId` for 3 s.

10.3 Test and Verification Procedure

The Phase 02 test validates the full USB-C → MQTT → CAM reception → use case detection round trip. The procedure uses the app as the receiver under test and an independent MQTT client (e.g. MQTT Explorer on a laptop connected to the same OBU via its Wi-Fi AP) as a verification observer, alongside a second CAM source (a second OBU, a test rig, or a simulated remote CAM publish) standing in for the approaching car or bike.

- Connect phone to OBU via USB-C cable. Enable USB tethering on the phone. Confirm the USB network interface is active (Settings; Connection screen shows "USB-C: Connected").
- Confirm MQTT broker connection. The Dashboard connection indicator shows "Connected" and `sys/state/heartbeat` messages appear in the V2X Monitor.
- On a laptop connected to the OBU via its Wi-Fi AP, open MQTT Explorer or run `mosquitto_sub` and subscribe to `v2x/rx/obu_gnss` and `v2x-uca/output/json/cam`.
- Confirm the OBU's own state appears on `v2x/rx/obu_gnss` at the expected rate (4 Hz) with `own_info.stationType = 2` (cyclist), on both the laptop subscriber and the app's V2X Monitor.
- Introduce a second CAM source representing an approaching car (real second OBU, test rig, or simulated remote CAM publish on `v2x-uca/output/json/cam` with a converging trajectory toward the ego position, sent as multiple successive samples so the app can derive a trend). Verify on the app's V2X Monitor that the CAM-based Use Case Alert panel (Section 10.4) raises an IMA-B alert at the correct alert level within the time thresholds of Section 5.5.

- Verify the alert clears automatically once the geometry resolves (the vehicles pass each other or diverge), and that no alert is raised for CAMs from road users that do not meet the IMA-B conflict geometry (e.g. a car travelling parallel, not crossing).
- Check OBU heartbeat (sys/state/heartbeat) throughout the test to confirm V2X stack health.

10.4 New and Updated UI Elements for Phase 02

- **Connection Setup screen:** Add USB-C connection status card (replaces the Wi-Fi card as the primary card). Wi-Fi card moves to the Developer section. USB-C card shows: cable state (connected/disconnected), detected OBU IP, and MQTT broker status.
- **V2X Monitor screen:** Add a CAM-based Use Case Alert panel showing: currently detected use case(s) (IMA-B now; RTW-B, LTW-B, IMA-S, SMVA/BCW-B as they come into scope), a live alert level badge (Info / Awareness / Warning, colour-coded per Section 5.5), and the remote station feeding the alert (station ID, distance, closing speed, time-to-conflict estimate). The raw message list shows the ego bike's own state (from v2x/rx/obu_gnss) and remote CAM entries (from v2x-uca/output/json/cam) as clearly distinct rows/sections; they arrive on different topics, not as "OWN"/"REMOTE" tags on a single stream.
- **Settings; Connection sub-screen:** Add USB-C section with: auto-detect toggle (on by default), manual IP override field, port field (default 1883). Wi-Fi section moved under Developer settings.
- **Settings; Use Case Alerts sub-screen:** Per-use-case enable/disable toggles (Section 2.2) and alert level threshold display, read-only for now.
- **Dashboard:** Transport indicator updated to show USB-C / Wi-Fi / BT with active transport highlighted. OBU GNSS fix quality (from v2x/rx/obu_gnss) displayed alongside phone GNSS quality for side-by-side comparison.

10.5 Phase 02 Key Technical Decisions

- **No raw ASN.1 encoding in the app (Phase 02 / CiT One only):** The app communicates with the OBU via the high-level consider it JSON MQTT API (processed messages). UPER ASN.1 encoding and decoding for CAM is handled entirely by the OBU's V2X stack. This keeps the app free of ASN.1 dependencies and ensures message compliance is enforced at the stack layer. **This principle is specific to the CiT One (Phase 02).** For the Phase 03 ESP32-C5 path, the phone takes over ASN.1 encoding/decoding directly (Section 13.1/13.5); the two OBU options have different ASN.1 boundaries by design.
- **Transport abstraction preserved:** The MQTT client layer (Eclipse Paho) does not change. Only the network interface bound to the broker's IP changes between USB-C, Wi-Fi, and Bluetooth phases. All MQTT topics, QoS levels, and subscription logic are identical across transports.
- **CAM as the sole V2X message type:** Per the C2C-CC Bicycle Safety White Paper [14], five of the six standardised bike-car use cases; including the project's primary IMA-B use case; rely exclusively on CAM/CAMv2. DENM generation and consumption are explicitly out of scope for the current project phase; only the Bike Accident Warning use case (BAW) requires DENM, and it is not part of the current scope (Section 1.2).
- **stationType for VRU:** OBU stationType must be cyclist (2). The microOBU's own CAMs are autonomously generated and must report stationType = cyclist (2) per ETSI EN 302 637-2 Table 1. This is a device-level OBU configuration, not an MQTT-settable value; the app can only read it back via own_info.stationType on v2x/rx/obu_gnss. The app should verify this on connection and alert the user if it does not read 2, as this affects VRU detection at equipped intersections.
- **Phase 02 success criteria:** (1) App connects to OBU MQTT broker over USB-C within 5 seconds of cable insertion. (2) App receives the OBU's own state on v2x/rx/obu_gnss at the expected rate (4 Hz) with own_info.stationType = 2 (cyclist). (3) When a remote CAM representing an approaching vehicle is received (real or simulated) on v2x-uca/output/json/cam across multiple successive samples, the app's Use Case Detection Engine correctly derives the

closing trend and raises an IMA-B alert at Awareness level within the time thresholds defined in Section 5.5. (4) The V2X Monitor clearly distinguishes the ego bike's own state (`obu_gnss`) from remote CAM entries (`v2x-uca/output/json/cam`). (5) No MQTT disconnections during a 5-minute continuous session.

11. Phase A; Trip Recording and Cyclist Event Detection

Phase A runs as a parallel development track alongside Phase 02 (OBU communication). It is independent of OBU connectivity; no cable, no MQTT broker, no hardware required. The objective is to detect meaningful cycling events (braking, turning, stopping) from phone sensors alone, record them during a trip, and store them locally for review. Once Phase 02 and Phase A are individually validated, a future phase will wire them together: detected events will feed into the CAM-based Use Case Detection Engine (Section 10.2) as corroborating signals, refining alert confidence and timing for the C2C-CC bike safety use cases in Section 1.2; entirely within the CAM-only scope of this project.

11.1 Orientation-Independent Sensor Strategy

The phone cannot be assumed to be mounted in a fixed orientation. It may be in a bag, jacket pocket, or handlebar mount at any angle. No axis-specific accelerometer reading (e.g. "X-axis = forward") can be relied upon. All event detection must therefore use orientation-independent signals:

- Total linear acceleration magnitude: $\sqrt{a_x^2 + a_y^2 + a_z^2} - 9.81 \text{ m/s}^2$. Removes gravity component; measures dynamic motion regardless of phone orientation.
- Total gyroscope magnitude: $\sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$. Total rotational energy of the device; turns and direction changes produce characteristic spikes regardless of mounting orientation.
- GPS speed (m/s): from FusedLocationProviderClient. Fully orientation-independent; primary signal for braking and stopping detection.
- GPS bearing change rate ($^\circ/\text{s}$): turning produces a sustained shift in GPS bearing above $\sim 2 \text{ m/s}$. Reliable even with phone in a bag.

A future improvement (Phase C+) will add a Kalman filter fusing GPS bearing, gyroscope integration, and magnetometer to produce a continuous, accurate heading estimate independent of GPS speed. This will enable directional acceleration decomposition (true forward/lateral deceleration) even at low speeds or indoors. This is not required for Phase A.

11.2 Running Standard Deviation Event Detector

During steady cycling, all magnitude signals have a low, stable variance. Events (braking, turning, stopping) manifest as a spike in standard deviation (sudden change onset) followed by a sustained shift in the running mean (the event itself). The detector monitors both simultaneously using a sliding window over the last N sensor samples.

Sliding window implementation:

- Window size: 50 samples at 50 Hz = 1 second of history. One RunningStats instance per signal (accel magnitude, gyro magnitude).
- Each new sample: add to circular buffer, evict oldest, recompute running mean and std dev in $O(1)$ using sum and sum-of-squares accumulators.
- Hysteresis on all thresholds: event fires when signal exceeds upper threshold for N consecutive frames (debounce); event clears only when signal drops below a lower threshold. Prevents rapid on/off flickering at the boundary.

Event detection rules (initial thresholds; to be tuned from real ride data):

- **BRAKING:** GPS speed decreasing by > 0.5 m/s from previous sample AND accel magnitude std dev > 1.2 m/s², sustained for > 300 ms (15 frames at 50 Hz). Confidence HIGH if speed drop > 1.5 m/s/s, MEDIUM otherwise.
- **TURNING:** Gyro magnitude running mean > 0.4 rad/s AND GPS bearing change rate $> 10^\circ$ /s (above 2 m/s), sustained for > 400 ms (20 frames). Confidence HIGH if both signals agree, LOW if only gyro triggers (possible phone movement in bag).
- **STOPPING:** GPS speed < 0.5 m/s sustained for > 2 s AND accel magnitude std dev < 0.15 m/s² (phone settled, not just GPS lag). Confidence HIGH when both conditions met.

11.3 Trip Recording Architecture

Sensor collection must continue when the app is backgrounded (phone in bag during a ride). This requires a foreground Android Service with a persistent notification. The ViewModel controls start/stop; the Service owns sensor lifetime and event writing.

- **TripRecordingService (Foreground Service):** Owns SensorManager registrations (accelerometer + gyroscope at 50 Hz) and FusedLocationProviderClient (1 Hz GPS). Feeds raw samples to EventDetector. Writes detected events and GPS track points to Room database via TripRepository. Displays a persistent notification showing recording state and elapsed time.
- **EventDetector:** Stateless signal processing class. Maintains one RunningStats sliding window per signal (accel magnitude, gyro magnitude). Applies event detection rules with hysteresis and debounce frame counts. Emits DetectedEvent objects via a Kotlin Flow. Has no Android dependencies; fully unit-testable.
- **TripRepository:** Room database abstraction. Writes RecordedTrip and DetectedEvent entities. Exposes trip history as a Flow<List<RecordedTrip>> for the trip list screen. GPS track stored as a sampled polyline (one point per second) serialised to JSON in the RecordedTrip row.
- **TripRecordingViewModel:** Binds the UI to TripRecordingService. Exposes StateFlow for recording state (IDLE / RECORDING / PAUSED), elapsed time, live event count, and current speed. Sends start/stop/pause intents to the Service.

11.4 Data Model

- **RecordedTrip:** id, startTime (epoch ms), endTime (epoch ms), distanceMetres, eventCount, gpsTrackJson (sampled polyline). One row per trip.
- **DetectedEvent:** id, tripId (FK), timestamp (epoch ms), type (BRAKING / TURNING / STOPPING), confidence (LOW / MEDIUM / HIGH), latitude, longitude, speedMps, peakAccelMagnitude, peakGyroMagnitude, durationMs. One row per detected event.
- **EventType enum:** BRAKING, TURNING, STOPPING. In a future phase, detected events will be correlated with the CAM-based Use Case Detection Engine (Section 10.2) as corroborating signals; for example, a BRAKING event coinciding with a detected SMVA/BCW-B alert condition increases confidence in that alert and can accelerate its escalation from Awareness to Warning level. This remains entirely CAM-based for the ESP32-C5 path; no DENM broadcast is triggered by these events (DENM remains CiT One-only, Section 13.4). In Phase 03 (Section 13), this detector's underlying turning/stopping state becomes a direct input to the phone's own outgoing CAM content, not just a locally logged event; see Section 12.7.

11.5 New UI Elements for Phase A

- **Trip Recording screen:** Large Record / Stop button. Live readouts: elapsed time, current GPS speed, event count this session (Braking: N, Turning: N, Stopping: N). Status indicator showing whether OBU is connected (if Phase 02 is also active) or standalone mode. Accessible from the bottom navigation bar.

- **Trip History screen:** List of past recorded trips, each showing date, duration, distance, and event count. Tap to open Trip Review.
- **Trip Review screen:** GPS track displayed on an OpenStreetMap tile layer. Detected events overlaid as coloured pins (red = braking, amber = turning, blue = stopping) at their GPS coordinates. Tap a pin to see event detail (type, confidence, speed, peak sensor values, timestamp). This screen is the primary tool for validating and tuning detection thresholds from real ride data.

11.6 Phase A Success Criteria

- TripRecordingService keeps sensor collection alive for a minimum 30-minute ride with the phone in a bag and the screen off, with no missed sensor samples.
- On a test ride with known deliberate events (3 hard brakes, 3 turns, 2 full stops), the detector records at least 7 out of 8 events at MEDIUM or HIGH confidence with no more than 2 false positives per 10 minutes of riding.
- Trip Review screen correctly shows the GPS track and event pins at the right locations for a known test route.
- EventDetector unit tests pass with synthetic sensor input covering all three event types, edge cases (very slow speed, standing still at start), and false-positive suppression (bag movement while stationary).

12. Future Architecture Considerations & Open Design Questions

This section records open engineering questions raised during Phase 02 design discussions, along with a proposed direction where one exists. None of these are Phase 02 requirements; they are documented here so they are not lost and can be revisited as the relevant phases come into scope.

12.1 Multi-Vehicle Handling & Notification Limits

Open question: with the per-station rolling history described in Section 10.2.1, how many concurrently-tracked remote vehicles can the Use Case Detection Engine handle, and; since several nearby cars could technically satisfy a use case's geometric conditions at once; how many should actually generate a user-facing notification? Uncontrolled, this risks notification spam, which directly conflicts with the "minimal interaction during use" and "clear feedback" UX principles (Section 5.3).

- Tracking capacity is not expected to be a performance concern: maintaining rolling history for tens of concurrent stations at typical CAM rates (1–10 Hz) is lightweight. The real constraint is the HMI, not the engine.
- **Proposed approach:** surface only the single highest-priority alert at a time; the remote vehicle with the shortest time-to-conflict, using alert severity (Section 5.5) as a tiebreaker. All other currently-tracked vehicles continue to be evaluated internally (so a new higher-priority alert can be surfaced immediately if conditions change) but do not generate a separate user-facing notification while a higher-priority one is active.
- This default should be treated as a starting point to validate and tune once real-world, multi-vehicle Phase 02 testing begins, rather than a final answer.

12.2 Intersection Ambiguity: Traffic-Light State (Future / Backlog)

Open question: at intersections, cars slow down for many reasons unrelated to the cyclist; a red light, congestion, an unrelated turn; and a closing-speed-based detector risks false positives in these situations. Fusing SPATEM/MAPEM signal-phase data (already defined message types, Glossary; published by an RSU where present) could let the engine distinguish "car slowing for a red light" from "car slowing because it is reacting to the bike."

This is explicitly logged as a **future/backlog item**, not near-term Phase 02 or Section 1.2 scope: it depends on RSU deployment and MAPEM/SPATEM coverage at the relevant test intersections (Section 1.1), which is not yet confirmed. Revisit once RSU availability at Kreuzung 1/2 (or later test sites) is known.

12.3 Shared-Road / Bike Lane Awareness Dataset

Most of Hamburg has dedicated cycling infrastructure, but bike-car conflict points cluster specifically where cyclists and vehicles share the road (unprotected lanes, driveway and side-street crossings, missing infrastructure segments). A future phase should incorporate a geospatial dataset of Hamburg's bike lanes and shared-road segments (e.g. derived from OpenStreetMap cycling infrastructure tags) so the detection engine can weight or tune alert sensitivity by context; for example, tighter thresholds at unprotected shared segments,

looser or suppressed alerting where cyclists and vehicles are physically separated. Not part of current scope; documented here as a future data dependency for the detection engine.

12.4 Geo-Server for Crowdsourced Trajectory Data

Future phase: a backend geo-server aggregating anonymised trajectory and event data from test rides (and eventually production use) to refine trajectory prediction and detection thresholds over time; a data-driven complement to the fixed thresholds in Sections 10.2.1 and 11.2. This would be the project's first requirement for a remote backend; Section 2.3 currently states none is required for Phase 01, and this remains true through Phase 02, but a geo-server is flagged here as a likely future requirement once enough ride data exists to make it worthwhile.

12.5 Geofencing Around High-Risk Intersections

Future phase: define geofences around the high-risk test intersections identified in Section 1.1 (Kreuzung 1/2) and any further locations identified from geo-server data (Section 12.4). If the geo-server is unreachable; no connectivity, which is expected to be common given the app's fully offline design (Section 3.2); the phone can still recognise, purely from its own GNSS position against a locally-cached geofence list, that it has entered a known high-risk area, and raise a lower-tier precautionary alert (e.g. Info level, Section 5.5) even without an active CAM-based use case detection. This acts as an offline fallback layer of awareness rather than a replacement for CAM-based detection.

12.6 Sensor Fusion Roadmap (Kalman Filter)

Reaffirming Section 11.1: a Kalman filter fusing GPS bearing, gyroscope integration, and magnetometer readings into a continuous, accurate heading and motion estimate remains the planned future direction, independent of raw GPS speed. In Phase 03 (Section 13.5), this fused estimate becomes the direct source of the phone-generated CAM's position, speed, heading, and yaw rate content; not just an input to the EventDetector as it is today.

12.7 EventDetector Evolution

Once the phone is responsible for generating CAM content (Section 13), the EventDetector's role expands beyond trip logging (Section 11): its continuous turning/stopping/braking classification; or, more likely, the underlying continuous heading and speed state feeding it once the Kalman filter (Section 12.6) is in place; becomes a direct input to what the phone reports as its own CAM kinematics, not just a locally logged event. The current threshold-based, discrete-event approach (Section 11.2) is expected to evolve alongside this: from flagging discrete events (braking/turning/stopping as onset/offset events) toward maintaining a continuous, always-current state estimate suitable for broadcasting, likely superseding or working alongside the discrete EventDetector rather than replacing its trip-logging function outright.

13. Phase 03 Overview; ESP32-C5 Dual-OBU & Phone-Generated CAM

Phase 03 formalises the future architecture direction first raised during Phase 02 design discussions (Section 12) into a concrete plan, centred on a specific second OBU option: the ESP32-C5. The phone becomes responsible for generating outgoing CAM content; the ESP32-C5 acts as a dumb ITS-G5 transceiver that transmits what the phone hands it and forwards received V2X frames back. The CiT One OBU remains fully supported and unchanged (Section 10); Phase 03 adds a choice, not a replacement.

13.1 ESP32-C5 as a Second OBU Option

The ESP32-C5 (Espressif Wi-Fi 6 / Bluetooth 5 / 802.15.4 SoC) is being evaluated as a low-cost, custom OBU alternative to the consider it CiT One. Unlike the CiT One, it does not run its own autonomous CAM generation or Use Case application layer, and; per the decision in Section 13.2/13.5; it does not perform UPER ASN.1 encoding/decoding either: it is a true "dumb" RF transceiver, sending and receiving raw encoded V2X frame bytes on the phone's instruction. **Open item:** the ESP32-C5's RF/PHY compliance with ITS-G5 (802.11p at 5.9 GHz) is still to be validated and is not assumed here; this is a hardware feasibility question for the team, not a software decision.

UPER ASN.1 encoder/decoder options for the phone (Kotlin): since the phone now owns ASN.1 encoding/decoding for this path (Section 13.2), it needs a UPER ASN.1 codec for the relevant V2X message types (CAM to start). Two options are being considered:

- **Port/reuse the consider it (CiT) compiler:** CiT already has a working ASN.1-to-code compiler for its V2X message set, but it is written in Rust. Using it would mean either translating its generated encode/decode logic to Kotlin by hand, or bridging the existing Rust code into the Android app directly (e.g. via JNI/UniFFI) rather than a full rewrite; the latter is likely lower-risk and less error-prone for a safety-critical wire format, and worth evaluating before committing to a manual Kotlin port.
- **Adopt an existing ASN.1 UPER encoder/decoder for V2X:** find and embed a ready-made ASN.1 UPER codec targeting the ITS-G5 message set (e.g. generated from the standard CAM/DENM ASN.1 schemas via a tool such as `asn1c` or `asn1scc`) directly into the phone app, rather than building one from scratch. Specific library choice and JVM/Android compatibility still need to be evaluated; not yet selected.

13.2 Transport: USB-C to ESP32 UART

Unlike the CiT One, which exposes its MQTT broker over USB-C via Android USB tethering (a virtual Ethernet interface, Section 10.1), the ESP32-C5 connects to the phone via USB-C into its UART port, i.e. a serial link rather than a network interface. This is an architecturally distinct transport path from the CiT One's USB-C/Ethernet-tethering approach, and needs its own connection handler in the app (Android USB Serial APIs rather than `ConnectivityManager/NetworkCallback`).

- **Open item:** the exact framing/protocol over the UART link (e.g. line-delimited JSON, a lightweight local MQTT-over-serial bridge, or a custom binary framing) is not yet defined and needs a design decision before implementation.

- The existing MQTT client abstraction (Section 4.2, Section 10.5) assumes a consistent MQTT-over-TCP interface across transports; the ESP32-C5's serial link may require either a phone-side MQTT-over-serial bridge to preserve that abstraction, or a parallel non-MQTT data path specific to the ESP32-C5. To be decided during implementation.

13.3 Settings: OBU Hardware Selection (CiT One / ESP32-C5)

A new OBU Hardware setting lets the user choose which physical OBU is connected: CiT One (Phase 02 behaviour, Section 10) or ESP32-C5 (Phase 03 behaviour, this section). This selection determines: which USB-C transport handler is used (Ethernet tethering vs. UART serial, Section 13.2), whether CAM is consumed from the OBU (CiT One) or generated by the phone and transmitted via the OBU (ESP32-C5, Section 13.5), and whether the DENM TX trigger is available (CiT One only, Section 13.4).

13.4 DENM Trigger Retained for CiT One Only

Earlier project versions of this document defined a DENM TX Control flow using the CiT One's high-level Use Case API: the app publishes a `uca-denmctrl` JSON control message (type, active, usecase, params, with the MQTT retain flag set) to topic `v2x-uca/input/denmtrg`, and the CiT One's Use Case app handles ETSI-compliant DENM parameter assembly, UPER ASN.1 encoding, and ITS-G5 broadcast. This project has since moved to a CAM-only detection approach for the app's own use cases (Section 0.2, Section 10.5), so this DENM trigger is not part of the core Phase 02 detection flow; but it remains available as a CiT One-specific capability, since the CiT One's own Use Case app already implements it.

- The DENM TX trigger (`v2x-uca/input/denmtrg`) is only offered in the UI when the CiT One is the selected OBU hardware (Section 13.3).
- It is **not available** when the ESP32-C5 is selected: the ESP32-C5 has no onboard Use Case app or DENM parameter assembly of its own, and although the phone will own ASN.1 encoding for the CAM path (Section 13.1/13.5), DENM message support is a separate scope decision, not a technical blocker; it has simply not been implemented for the ESP32-C5 path yet and remains out of scope for Phase 03 unless separately prioritized.

13.5 Phone-Generated CAM (ESP32-C5 Path)

When the ESP32-C5 is the selected OBU, CAM content is generated entirely on the phone and handed to the ESP32-C5 for transmission, rather than the OBU generating it autonomously (as the CiT One does, Section 10.2).

- **Position/motion source:** the phone's own GNSS and IMU are fused and "baked into" the outgoing CAM's kinematic fields (position, speed, heading, and; once available; yaw rate/acceleration). There is no separate OBU GNSS in this path; unlike the CiT One path (Section 10.2.1), there is no OBU-primary/phone-fallback distinction to make, since the phone's own sensors are the only source.
- **ASN.1 boundary (reversed from the initial Phase 03 direction):** the phone now performs UPER ASN.1 encoding of outgoing CAM content and decoding of received V2X frames itself (Section 13.1), rather than handing structured fields to the hardware. The ESP32-C5 becomes a true "dumb" RF transceiver, sending and receiving raw encoded bytes. **Working assumption, not yet confirmed:** the ESP32-C5 is expected to still handle the lower layers below ASN.1; GeoNetworking/BTP framing and the ITS-G5 RF/PHY/MAC; with the phone responsible only for the ASN.1-encoded application payload; the exact split of GeoNetworking/BTP responsibility

between phone and hardware is an open item for the team to confirm. This differs from the Phase 02 architecture (Section 10.5), where the CiT One's onboard V2X stack owns ASN.1 encoding end-to-end.

- **Future refinement:** the Kalman filter planned in Section 11.1/12.6 is the intended long-term source of the fused position/heading estimate feeding this CAM content; until then, the phone's raw fused GNSS + IMU state is used directly.

13.6 Adaptive CAM Transmission Rate

Phone-generated CAM transmission (Section 13.5) is gated by an active Recording Session (Section 2.1): CAM is only sent while a driving session is being recorded, not continuously in the background.

- **Base rate:** 1 Hz once a recording session has started.
- **Adaptive increase:** the rate increases above 1 Hz when the rider is in a known high-risk area; reusing the geofencing concept from Section 12.5 (high-risk intersections, e.g. Kreuzung 1/2 and any further locations identified via the geo-server, Section 12.4) as the trigger condition. Exact target rate(s) and the geofence radius/detection logic are not yet defined; to be tuned once the geofence list and real ride data are available.
- **Open item:** whether additional conditions beyond geofenced risk areas (e.g. detected nearby traffic density from received remote CAMs, or an active use-case alert per Section 10.2) should also drive the rate upward is not yet decided.

13.7 UI Updates: V2X Monitor Map View & Dashboard Overview

- **V2X Monitor map view:** the screen gains a live map (reusing the OpenStreetMap tile layer approach already planned for the Trip Review screen, Section 11.5) plotting the ego bike's own GNSS position and the positions of detected remote vehicles from CAM data, as an alternative to reading raw per-topic JSON. The existing raw message list (Section 10.4) is retained alongside the map, not replaced by it.
- **Dashboard redesign:** reorganised into sensor overview cards; GNSS, IMU, and OBU connection status (CiT One or ESP32-C5, whichever is selected, Section 13.3); at a glance, plus a prominent "Start Driving Session" shortcut that navigates directly to the Recording Session screen, lowering friction to begin a session (consistent with the "Low friction setup" UX principle, Section 5.3).

13.8 Detection Engine: Unchanged for Reception

The CAM-based Use Case Detection Engine (Section 10.2, Section 10.2.1) is unchanged on the receive side regardless of which OBU is connected. Whether remote CAMs arrive via the CiT One (Section 10.1) or the ESP32-C5 (Section 13.2), they reach the same v2x-uca/output/json/cam-equivalent processing path and the same detection logic runs on top of them. Only the source of the ego bike's own outgoing CAM (Section 13.5) and the transport layer (Section 13.2) differ by selected hardware.

13.9 Bluetooth Transport (Open Discussion)

Bluetooth as a production transport; previously deferred in favour of USB-C for Phase 02 (Section 8); is back under active discussion now that the ESP32-C5 (which supports Bluetooth 5) is being evaluated as an OBU option. No decision has been made yet on whether Bluetooth becomes the production transport for the ESP32-C5 path, the CiT One path, both, or neither. This needs a dedicated discussion covering at minimum: target connection setup time (≤ 2 s per the existing project spec, Section 8.1), power consumption

trade-offs versus USB-C, and whether it replaces or supplements USB-C in production builds.

References

All documents, standards, libraries, and hardware referenced during the development of this requirements specification.

Ref	Citation
Project Documents	
[1]	consider it GmbH. V2X Unit MQTT APIs; BU Consider Innovation. Document v6 (rev. 9b55730), 2025-02-21. CI-CiT-MQTT_API_Documentation-v6-20250221.pdf.
[2]	consider it GmbH / HAW Hamburg. Projektbeschreibung micrOBU v2; Appendix to Annex 4, Project Description, ZIM Funding Program, BMWK. 2024-12-12. (241212_Projektbeschreibung_micrOBU_v2 en-US.pdf)
[3]	HAW Hamburg. V2X OBU Companion App Requirements & Documentation. This document. V2X_MicrOBU_Android_App_Requirements_v15.docx.
MQTT	
[4]	Eclipse Foundation. Eclipse Paho Android Client. MQTT 3.1.1 client library for Android. https://github.com/eclipse/paho.mqtt.android
[5]	Eclipse Mosquitto Project. Mosquitto; An Open Source MQTT Broker. Version 2.0.11 (deployed on micrOBU). https://mosquitto.org
[6]	OASIS Standard. MQTT Version 3.1.1. OASIS Standard, 29 October 2014. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
V2X / ITS Standards	
[7]	ETSI EN 302 637-2 V1.4.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service. ETSI, 2019.
[8]	ETSI EN 302 637-3 V1.3.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service (DENM). ETSI, 2019. (Background standard; DENM is not implemented in the current app scope, see Section 0.2.)
[9]	ETSI TS 103 301 V2.1.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Facilities layer protocols and communication requirements for infrastructure services (MAPEM, SPATEM, SREM, SSEM, IVIM). ETSI, 2020.
[10]	ETSI EN 302 636-4-1 V1.4.1. Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications. ETSI, 2020.

[11]	IEEE Std 802.11p-2010. IEEE Standard for Information Technology; Telecommunications and Information Exchange Between Systems; Local and Metropolitan Area Networks; Specific Requirements. Part 11: Wireless LAN MAC and PHY Specifications. Amendment 6: Wireless Access in Vehicular Environments. IEEE, 2010.
[12]	ETSI TS 103 097 V2.1.1. Intelligent Transport Systems (ITS); Security; Security header and certificate formats. ETSI, 2021. (Referenced for signed V2X message format and PKI requirements.)
C2C-CC / Bicycle Safety Use Cases	
[13]	CAR 2 CAR Communication Consortium. Basic System Profile. Release 1.6.8, December 2025. Available: https://www.car-2-car.org/fileadmin/documents/Basic_System_Profile/Release_1.6.8/C2CCC_RS_2037_Profile_R168.pdf
[14]	CAR 2 CAR Communication Consortium. White Paper on Bicycle Safety Use Cases. Document 2324, v1.0, 2026-05-07. C2CCC_WP_2324_Bicycle_SafetyUseCases_V1.0.
Custom Hardware (Phase 03)	
[16]	Espressif Systems. ESP32-C5 Series Datasheet. Wi-Fi 6 / Bluetooth 5 (LE) / IEEE 802.15.4 SoC. Available: https://www.espressif.com/en/products/socs/esp32-c5 (ITS-G5 / 802.11p compliance not covered by this datasheet; to be independently validated, Section 13.1).
Android & Kotlin Libraries	
[15]	Google LLC. Jetpack Compose; Android's modern toolkit for building native UI. https://developer.android.com/jetpack/compose
[22]	Roger Light et al. Eclipse Mosquitto; Open Source MQTT Broker. Debian package mosquitto 2.0.11-1+deb11u2 (arm64). Deployed on micrOBU dev unit at 192.168.3.202:1883.